50325-0868 (Seq. No. 8749) *Patent*

UNITED STATES PATENT APPLICATION

FOR

PREVENTING NETWORK RESET DENIAL OF SERVICE ATTACKS

INVENTORS:

MITESH DALAL
AMOL KHARE
RANDALL STEWART

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CA 95125
(408) 414-1080

EXPRESS MAIL MAILING INFORMATION

"Express Mail" mailing label number: EV323351215US

Date of Deposit: January 9, 2004

PREVENTING NETWORK RESET DENIAL OF SERVICE ATTACKS

FIELD OF THE INVENTION

[0001] The present invention generally relates to computer networks. The invention relates more specifically to preventing TCP Reset attacks in networks.

BACKGROUND

[0002] The approaches described in this section could be pursued, but are not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated herein, the approaches described in this section are not prior art to the claims in this application and are not admitted to be prior art by inclusion in this section.

[0003] Networks have become an important tool for businesses and consumers alike, many of which are now dependent on the constant availability of network resources such as mail servers, Web sites, and content servers. As use of networks increases, protecting networks from disruption by malicious entities through denial of service ("DoS") attacks becomes more critical. DoS attacks deprive legitimate users of access to network services, and have been used successfully to disrupt legitimate user access to internet sites such as Yahoo! and CNN.

[0004] One type of DoS attack takes advantage of the basic design of the Transmission Control Protocol ("TCP"), one of the foundational protocols of the Internet, as defined in Internet Engineering Task Force (IETF) Request for Comments (RFC) 793. This type of DoS attack, known as a TCP Reset or RST attack, exploits the fact that under an implementation of TCP that is compliant with RFC 793 a TCP connection can be properly terminated in response to receiving a TCP packet from a remote node in which the Reset bit ("RST bit") in the header is set.

[0005] A TCP Reset attack seeks to shut down legitimate TCP sessions by injecting, into an active TCP connection, a spoofed segment with the Reset (RST) flag set and containing a packet sequence value that falls within a range of valid sequence values as allowed by the receiving node. Typically an attacker first determines or guesses the IP addresses of both endpoints and the port numbers that the endpoints are using for a TCP connection or higher-level protocol. A successful attacker also guesses a sequence number that falls within the allowed range or window. Sending any TCP segment with the RST flag set, correct IP addresses and port numbers with the sequence number falling within the window of the TCP connection can cause a receiving node that implements TCP properly under RFC 793 to shut down the TCP connection.

[0006] A TCP SYN attack proceeds in a similar manner. If an attacker sends a TCP packet with the SYN bit set in the header, and a sequence value that falls within a window of allowed sequence values, under RFC 793 the receiving node closes the TCP connection and sends a TCP RST packet. RFC 793 provided this process in order to enable non-synchronized hosts to close a connection and re-synchronize, but in contemporary practice the process creates security vulnerability.

[0007] Border Gateway Protocol (BGP), Hypertext Transfer Protocol (HTTP), some voice protocols, Multi-Protocol Label Switching (MPLS), and other protocols use TCP connections and are targets for these attacks. The consequences can be severe. For example, when a BGP session of a router is disrupted by closing the associated TCP connection, the router will discard all BGP routes that it has created, essentially causing a failure of the BGP process. As a result, the BGP process must re-synchronize itself with peer routers in the network, and during the re-synchronization period the failed router cannot forward any traffic. Accordingly, researchers in this field are interested in creating ways to thwart TCP Reset attacks, without fundamentally altering the operation of TCP as specified in RFC 793.

[0008] In one approach, researchers have thought that by assigning a pseudo-random 32-bit value as the Initial Sequence Number (ISN) for a new TCP connection, an attacker could not guess the correct sequence number in any practical way, because the number of potentially correct values is 232 or approximately 4 billion values, making such an attack virtually impossible. This principle may be true in the case of an attacker who attempts to inject a data segment into an existing TCP connection.

[0009] However, a conventional TCP implementation compliant with RFC 793 will accept a RST segment or SYN packet if the sequence number of the segment falls within a window or range of acceptable values, even if the sequence number is not an exact match to the next expected sequence number. This approach is used to compensate for the possibility that packets may be lost. In some implementations of TCP the range of allowed sequence values may be as large as 16,000 to more than 50,000 values. Unfortunately, a consequence is that the attacker does not need to generate all 32 bits of the sequence number correctly to provide a number that a receiving node will accept, even when a truly random or pseudorandom ISN is used. If the range of allowed sequence values is sufficiently large, then the chance is greatly increased that an attacker can guess a correct sequence value through either random or brute-force selection in a practical amount of time. The larger the window established by the receiving node, the easier it is for the hacker to carry out this attack.

[0010] Thus, a TCP implementation that merely checks a newly arrived RST packet or SYN packet for an established connection to determine whether the sequence number is within a given window, and tears down the connection if so, is inadequate to prevent attack by a hacker seeking to terminate a connection prematurely.

[0011] Another approach, as used in the OpenBSD implementation of TCP under UNIX, guards against a Reset attack by requiring that the Reset packet carry a sequence number that is exactly the next expected sequence number, and not just within the expected window. If a Reset packet carries a sequence number that is not an exact match, the TCP process ignores the Reset packet and does nothing. However, this approach is considered impractical, because for a receiver more often than not the sequence number of a TCP packet is not the same as the next expected due to loss of packets. When lost packets occur in the OpenBSD approach, the receiving node is left with a connection that the sending node considers closed.

[0012] In still another approach, TCP RST "damping" is performed in which a node ignores RST packets if too many such packets arrive within a given time period. However, this approach is inadequate because even one improper RST packet may be sufficient to induce closure of a critical TCP connection.

[0013] An approach for addressing a similar attack, known as the SYN-RST attack, is provided in co-pending application Ser. No. 10/641,494, filed August 14, 2003, entitled "Detecting network denial of service attacks," of Pritam Shah et al., and assigned to the same assignee hereof. The approach of Shah et al. is appropriate for an intermediate router rather than a TCP endpoint device, but does not fully address all issues described in this disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0015] FIG. 1 is a state diagram that illustrates one embodiment of a process for preventing TCP RST attacks;

[0016] FIG. 2A is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 1 when an attack is underway;

[0017] FIG. 2B is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 1 when no attack is occurring;

[0018] FIG. 3 is a state diagram that illustrates another embodiment of a process for preventing TCP RST attacks;

[0019] FIG. 4A is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 3 when an attack is underway;

[0020] FIG. 4B is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 3 when no attack is occurring;

[0021] FIG. 5 is a state diagram that illustrates one embodiment of a process for preventing TCP SYN attacks;

[0022] FIG. 6A is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 5 when an attack is underway;

[0023] FIG. 6B is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 5 when no attack is occurring; and

[0024] FIG. 7 is a block diagram that illustrates a computer system upon which an embodiment may be implemented.

DETAILED DESCRIPTION

[0025] A method and apparatus for preventing network denial of service attacks is described. In the following description, for the purposes of explanation, numerous specific details are set forth to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form to avoid unnecessarily obscuring the present invention.

[0026] Embodiments are described herein according to the following outline:

1.0 GENERAL OVERVIEW

[0027] As an introduction only, and without limiting the scope of the appended claims, the needs identified in the foregoing Background, and other needs and objects that will become apparent for the following description, are achieved in the present invention, which comprises, in one aspect, a method for preventing an attack on a network, wherein the attack comprises sending a spurious transmission control protocol (TCP) packet with a Reset (RST) bit set, the method comprising the computer-implemented steps of receiving, from a remote end node, a packet of a flow in which a RST bit of a TCP header is set; determining whether a sequence value in the packet is within a range of allowed sequence values; and when the first sequence value is within the range of allowed sequence values, sending an acknowledgment message without closing a TCP connection associated with the flow.

[0028] According to one feature of this aspect, the method further involves receiving, from the remote end node, a next packet of a flow in which the RST bit is set and comprising a second sequence value; determining whether the second sequence value is equal to an expected sequence value; and closing a TCP connection associated with the flow only when the second sequence value is equal to the expected sequence value.

[0029] In another aspect, a method of preventing an attack on a network is provided, comprising the computer-implemented steps of receiving, from a remote end node, a packet of a flow in which a RST bit of a TCP header is set; and sending an acknowledgment message without closing a TCP connection associated with the flow and without regard to whether a sequence value in the packet is within a range of allowed sequence values. The method may further involve setting a flag representing receipt of the packet with RST bit set; receiving, from the remote end node, a next packet of a flow in which the RST bit is set and comprising a sequence value; determining whether the sequence value equals an expected sequence value; and closing the TCP connection only when the sequence value equals the expected sequence value.

[0030] In yet another aspect, a method of preventing an attack on a network is provided comprising the steps of receiving, from a remote end node, a packet of a flow in which a SYN bit of a header is set; sending an acknowledgment message without closing a TCP connection associated with the flow and without regard to whether a sequence value in the packet is within a range of allowed sequence values; receiving a next packet of the flow; and when the next packet is a TCP RST packet, performing the steps of either of the other two aspects described above with respect to the next packet.

[0031] In other aspects, the invention encompasses a computer apparatus and a computer-readable medium configured to carry out the foregoing steps. Further, many other features and aspects will become apparent from the following description and from the appended claims.

2.0 APPROACHES FOR PREVENTING TCP DENIAL OF SERVICE ATTACKS

[0032] Two approaches for preventing TCP RST denial of service attacks, and one approach for preventing TCP SYN attacks, are presented. In a first approach for RST attacks, when a RST arrives and does not contain the next expected sequence value but the sequence value is within the correct window of data, a receiver node sends an acknowledgement but does not close the connection. If a legitimate sending node has closed the connection, the remote end node will generate a RST in reply to the acknowledgement, and near end node can properly close the connection.

[0033] A second approach operates in a similar manner, but by default all RST packets are invalid. On receipt of a RST packet, including a packet that carries the expected sequence value, the receiver node automatically sends an acknowledgement back and sets a flag indicating it has seen a RST packet. If the receiver node next receives is a RST packet with the correct sequence number, then the receiver node closes the connection. If the next segment it sees is not a RST packet, then the flag is cleared.

[0034] Accordingly, on receipt of a RST packet, a receiver node challenges the peer node in the then-current connection by means of an acknowledgement message. The peer node either responds with a RST packet as a response if the connection is genuinely closed, or the peer node takes no action if the connection remains open. The disclosed approaches provide protection from RST DOS attacks, and provide faster and more deterministic connection termination in response to receiving RST packets. The approaches are typically applied at an endpoint node for TCP connections that are in the ESTABLISHED or synchronized state as defined in RFC 793.

[0035] For TCP RST attack prevention, a node may implement only one of the approaches herein, or may implement both approaches and enable a user to select one or the other according to a perceived threat level. Generally, the second approach is more appropriate for a higher perceived threat level.

2.1 FIRST APPROACH FOR PREVENTING TCP RST ATTACKS

[0036] A first approach for preventing TCP RST attacks is now described with reference to FIG. 1, FIG. 2A, and FIG. 2B. FIG. 1 is a state diagram that illustrates one embodiment of a process for preventing TCP RST attacks; FIG. 2A is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 1 when an attack is underway; FIG. 2B is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 1 when no attack is occurring.

[0037] The process of FIG. 1 may be implemented in one or more computer programs, sequences of instructions or other software elements that are executed by a network element acting as an endpoint in a TCP connection. For example, FIG. 1 may be implemented as part of a TCP application or feature of an operating system of a router, switch or other element of network infrastructure. In FIG. 2A and FIG. 2B, network elements or nodes R1, R2 represent endpoints in a TCP connection.

[0038] Nodes R1, R2 may comprise routers, switches, hubs, gateways, personal computers, workstations, servers, or other devices that are or can be connected to or communicate with a network. Attacker 202 is any entity that is injecting unwanted or spurious segments or packets into a TCP flow that has been established between nodes R1 and R2. Attacker 202 may comprise a workstation, personal computer, router, switch, or other processing element.

[0039] Nodes R1, R2, and Attacker 202 participate in one or more networks. Further, nodes R1, R2 and Attacker 202 may be in or accessible through a local area network (LAN), wide area network (WAN), one or more internetworks, or any other kind of network or subset thereof, in which the Transmission Control Protocol (TCP) is used to establish connections between network elements. Such a network may contain additional network infrastructure elements such as routers, switches, etc. and other end station devices such as workstations, printers, servers, etc. In one implementation, nodes R1, R2 and Attacker 202 all are communicatively coupled to a public packet-switched network such as the internet.

[0040] Nodes R1, R2 and Attacker 202 may be connected to additional network elements. The system shown in FIG. 2A, 2B is just one of many possible different configurations. Other embodiments may include fewer or more system components than those illustrated. Specifically, in a practical system there may be any number of network elements.

[0041] For purposes of describing FIG. 1, assume that R2 of FIG. 2A, FIG. 2B has implemented the process of FIG. 1; node R1 may implement the same process, or may implement TCP only as defined in RFC 793.

[0042] Initially, an element that implements FIG. 1 is in an idle state 101, which broadly represents any state of relative inactivity of a thread, process or other element that implements or embodies a TCP connection. A transition to Perform Handshake 102 occurs when a TCP SYN packet is received, without the Acknowledge (ACK) or Reset (RST) bits set, to initiate handshaking for a new connection. If handshaking is successful, a transition to Normal TCP Data Communication state 104 occurs. The details of handshaking and establishing a connection, and processing in response to unsuccessful handshaking, are not pertinent to this disclosure, but are defined in RFC 793.

[0043] At state 104, data is exchanged among the endpoint nodes. For example, nodes R1, R2 communicate data as indicated by arrow 1 in FIG. 1. A TCP connection between nodes R1, R2 is in the ESTABLISHED or synchronized state at state 104.

[0044] Upon receiving a TCP packet with the RST bit set, a transition to Evaluate Sequence Value state 106 occurs. In state 106, a sequence value carried in the TCP header is evaluated and one of several transitions occurs depending on whether the sequence value is equal to an expected next sequence value, not equal to the expected next sequence value but within a range of allowed next sequence values, or not equal to the expected next sequence value and not within the range of allowed next sequence values. A TCP packet with the RST bit set could be received from Attacker 202 in the attack scenario of FIG. 2A, as indicated by arrow 2, or from a legitimate endpoint such as R2, as shown by arrow 2 in FIG. 2B.

[0045] If the sequence value is equal to an expected next sequence value, then the TCP connection is closed, as indicated by Close Connection state 108. Because the range of all possible sequence values is 232, an attacker is exceedingly unlikely to guess the exact expected next sequence value. Therefore, a RST packet that includes an exactly correct next sequence value is assumed to be a legitimate RST packet that requires the connection to be closed. A transition to Idle state 101 then occurs. For example, a thread or process representing the then-current TCP connection may be terminated and associated resources may be reclaimed.

[0046] If the sequence value is not equal to the expected next sequence value but within a range of allowed next sequence values, then an acknowledgment message is sent, as indicated in Send Acknowledge state 110. A TCP packet with ACK bit set in the header and carrying the next expected sequence value may be used. Sending the acknowledge message in an attack scenario is represented as arrow 3 in FIG. 2A; sending the acknowledge message in a normal scenario is shown as arrow 3 in FIG. 2B. A transition to Normal state 104 then occurs.

[0047] In effect the acknowledge message sent at state 110 acts as a challenge to the other endpoint to prove knowledge of the exact next expected sequence number if that endpoint legitimately intends to close the connection. Thus, in the attack scenario of FIG. 2A, node R1 takes no action in reply to the acknowledge message of arrow 3 because it did not originate the RST packet of arrow 2. Node R1 need not take action under RFC 793 because an unsolicited acknowledgement message may be ignored. As a result, the attack by Attacker 202 is prevented.

[0048] In contrast, in the normal scenario of FIG. 2B, node R1 issues a further RST packet at arrow 4 that will include the exact next expected sequence value that node R1 received in the ACK message of arrow 3. In response, node R1 transitions to Close Connection state 108, in which node R1 properly closes the connection.

[0049] In state 106 if the sequence value is not equal to the expected next sequence value and not within the range of allowed next sequence values, then the packet is dropped, as Drop Packet state 112 indicates. Optionally, further responsive action may be taken, as shown by Responsive Action state 114. Such responsive action may include accumulating a counter that counts spurious RST packets until a specified threshold is reached, generating a notification message to a network operations center, publishing an event on an event bus, etc.

[0050] 2.2 SECOND APPROACH FOR PREVENTING TCP RST ATTACKS

[0051] A second approach for preventing TCP RST attacks is now described with reference to FIG. 3, FIG. 4A, and FIG. 4B. FIG. 3 is a state diagram that illustrates another embodiment of a process for preventing TCP RST attacks; FIG. 4A is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 3 when an attack is underway; and FIG. 4B is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 3 when no attack is occurring.

[0052] Referring first to FIG. 3, a connection initially is in Idle state 101, and proceeds to Perform Handshake state 102 and Normal Data Communication state 104 as described above for FIG. 1. The TCP connection is in the ESTABLISHED or synchronized state. Assume that a TCP RST packet is then received. In response, a transition to Test Flag state 302 occurs, in which a flag indicating prior receipt of a RST packet is tested. If the flag is not set, then a transition to Set Flag state 304 occurs, in which the flag is set. In one embodiment, the flag is set only if the sequence value carried in the RST packet is within the range of allowed values. Alternatively, the flag is set without regard to whether the sequence value is within the allowed range.

[0053] An acknowledgement message is then sent back to the remote end node, as indicated by Send Acknowledge state 306. The acknowledgment message of state 306 is sent regardless of whether a sequence value contained in the TCP RST packet is equal to an expected next sequence value or within a window of allowed sequence values. A transition to Await Next Packet state 307 then occurs.

[0054] In Await Next Packet state 307 the process awaits receipt of the next TCP packet. If the next packet is a RST packet, then a transition to Evaluate Sequence Value state 308 occurs. The sequence value within the RST packet is examined. If the sequence value exactly matches a next expected sequence value, then the connection is closed, as shown by Close Connection state 108.

[0055] If the sequence value does not exactly match the next expected sequence value, then the packet is dropped in Drop Packet state 112, leaving the TCP connection unaffected. No consideration is given to whether the sequence value is within an allowed window because a legitimate remote end node always will reply to an acknowledge message with the expected sequence value. Optionally, in Responsive Action state 114, other responsive action may be taken, as described above with respect to FIG. 1.

[0056] If the next TCP packet received at Await Next Packet state 307 is not a RST packet or is a RST packet with an incorrect sequence number, then the flag value is cleared, as indicated in Clear Flag state 310. A transition to Normal Data Communication state 104 then occurs.

[0057] Operation of the foregoing process is now described in the context of an attack and in the context of a legitimate closing of a connection.

[0058] Referring to FIG. 4A, assume that nodes R1 and R2 have established normal TCP communication as indicated by arrow 1, and Attacker 202 then sends a spurious RST packet, as indicated by arrow 2. In response, using the process of FIG. 3, receiving endpoint node R2 sets the flag value at Set Flag state 304 and as indicated by numeral 3 of FIG. 4A. Node R2 sends an acknowledge message, as shown by arrow 4 and as indicated by Send Acknowledge state 306. Because node R1 has not closed the connection, node R1 treats the acknowledge message as an unsolicited acknowledgement that requires no response. Accordingly, node R1 continues sending data in normal TCP communication, as indicated by arrow 5. As a result, the attack by Attacker 202 is thwarted.

[0059] In the normal scenario of FIG. 4B, nodes R1 and R2 again establish normal TCP communication as indicated by arrow 1, but node R1 then closes the connection and sends a legitimate RST packet to receiving node R2. In response, node R2 sets the flag value (numeral 3) and sends an acknowledge message (arrow 4) that contains a next expected sequence value.

[0060] Because node R1 has closed the connection, node R1 responds with another RST packet that contains the expected sequence value. As a result, R2 transitions to state 308, state 108, and state 310 of FIG. 3, properly closing the connection and clearing the flag value, as indicated by numeral 6 and numeral 7 of FIG. 4B.

2.3 APPROACH FOR PREVENTING TCP SYN ATTACKS

[0061] FIG. 5 is a state diagram that illustrates one embodiment of a process for preventing TCP SYN attacks; FIG. 6A is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 5 when an attack is underway; and FIG. 6B is a message flow diagram that illustrates TCP messages exchanged by nodes using the process of FIG. 5 when no attack is occurring.

[0062] Referring first to FIG. 5, from Idle state 101 a connection is established through Perform Handshake state 102 and normal TCP communication between first and second endpoint nodes occurs at Normal Data Communication state 104. The TCP connection is in the ESTABLISHED or synchronized state. Assume that a first endpoint node then receives a TCP packet with the SYN bit set in the header. A transition occurs to Send Acknowledge state 502, in which the first endpoint node sends an acknowledge message to the second endpoint. The acknowledgment message of state 502 is sent regardless of whether a sequence value contained in the TCP SYN packet is equal to an expected next sequence value or within a window of allowed next sequence values.

[0063] In contrast, under a conventional implementation of RFC 793, receiving a TCP SYN packet causes a receiving node to send a TCP RST packet if the sequence value of the TCP SYN packet is within the window of allowed next sequence values.

[0064] A transition then occurs directly to Await Next Packet state 504, in which the first endpoint node awaits receipt of the next TCP packet.

[0065] If the next packet is not a RST packet, then a transition to Normal Data Communication state 104 occurs and no action is taken with respect to the connection. Such a transition would occur, for example, if the sending node has not closed the connection and treated the acknowledgment of state 502 as an unsolicited acknowledgment that requires no reply.

[0066] If the next packet is a RST packet, then optionally either the first approach or second approach described above for preventing RST packets is initiated. Thus, at state 506, the first endpoint node enters either state 106 of FIG. 1 or state 302 of FIG. 3. In this manner, the RST packet that was just received is subjected to the same scrutiny for attack as provided in the processes of FIG. 1 or FIG. 3. Performing such additional checks is not required, but could provide additional assurance that the RST is legitimate.

[0067] In still another alternative, the processes of FIG. 1 or FIG. 3 may be performed only if the TCP RST packet received while in state 504 contains a sequence value that is not equal to the expected next sequence value. If a TCP RST packet is received at state 504 that contains a sequence value that is equal to the expected next sequence value, then the connection may be closed, because most likely, only an endpoint node that actually closed the connection has sent such a packet.

[0068] The process of FIG. 5 may operate as follows in the attack scenario of FIG. 6A and the normal scenario of FIG. 6B. Referring first to FIG. 6A, assume that nodes R1 and R2 as TCP endpoint nodes establish normal TCP communication as shown by arrow 1. Attacker 202 then sends a spurious TCP SYN packet to node R2 as shown by arrow 2.

[0069] In response, as indicated by arrow 3 and as shown in Send Acknowledge state 502, R2 sends an acknowledge message and enters Await Next Packet state 504. Because node R1 has not closed the connection, node R1 treats the acknowledge message as an unsolicited acknowledgement that requires no response. Therefore, node R1 takes no action with respect to the connection.

[0070] At some point thereafter node R1 sends further data in TCP packets. At state 504 the node R2 determines that the next packet is not a RST packet, and therefore transitions to Normal Data Communication state 104. As a result, the attack by Attacker 202 is prevented.

[0071] In the normal scenario of FIG. 6B, nodes R1 and R2 establish a TCP connection at arrow 1 and transfer data. Node R1 then properly closes the connection and sends a TCP SYN packet as shown by arrow 2. In response, node R2 sends an acknowledge message at arrow 3 as provided by Send Acknowledge state 502, and enters Await Next Packet state 504.

[0072] Because node R1 has closed the connection, node R1 replies with a RST packet at arrow 4. In response, at state 504 node R2 determines that the next packet is an RST packet and transitions to state 506. Node R2 then evaluates the RST packet according to the approach of either FIG. 1 or FIG. 3, as indicated by numeral 5 of FIG. 6B.

[0073] Accordingly, the RST packet undergoes further evaluation to determine whether it represents an attack. Further message flows as shown in FIG. 2B, FIG. 4B are performed in which node R2 sends another acknowledge message as a challenge and node R1 replies with the correct expected sequence number. The connection is then closed properly.

3.0 IMPLEMENTATION MECHANISMS -- HARDWARE OVERVIEW

[0074] FIG. 7 is a block diagram that illustrates a computer system 700 upon which an embodiment may be implemented. The preferred embodiment is implemented using one or more computer programs running on a network element such as a router device. Thus, in this embodiment, the computer system 700 is a router.

[0075] Computer system 700 includes a bus 702 or other communication mechanism for communicating information, and a processor 704 coupled with bus 702 for processing information. Computer system 700 also includes a main memory 706, such as a random access memory (RAM), flash memory, or other dynamic storage device, coupled to bus 702 for storing information and instructions to be executed by processor 704. Main memory 706 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 704. Computer system 700 further includes a read only memory (ROM) 708 or other static storage device coupled to bus 702 for storing static information and instructions for processor 704. A storage device 710, such as a magnetic disk, flash memory or optical disk, is provided and coupled to bus 702 for storing information and instructions.

[0076] A communication interface 718 may be coupled to bus 702 for communicating information and command selections to processor 704. Interface 718 is a conventional serial interface such as an RS-232 or RS-422 interface. An external terminal 712 or other computer system connects to the computer system 700 and provides commands to it using the interface 714. Firmware or software running in the computer system 700 provides a terminal interface or character-based command interface so that external commands can be given to the computer system.

[0077] A switching system 716 is coupled to bus 702 and has an input interface 714 and an output interface 719 to one or more external network elements. The external network elements may include a local network 722 coupled to one or more hosts 724, or a global network such as Internet 728 having one or more servers 730. The switching system 716 switches information traffic arriving on input interface 714 to output interface 719 according to pre-determined protocols and conventions that are well known. For example, switching system 716, in cooperation with processor 704, can determine a destination of a packet of data arriving on input interface 714 and send it to the correct destination using output interface 719. The destinations may include host 724, server 730, other end stations, or other routing and switching devices in local network 722 or Internet 728.

[0078] The invention is related to the use of computer system 700 for implementing the approaches herein. According to one embodiment of the invention, an implementation of the approaches herein is provided by computer system 700 in response to processor 704 executing one or more sequences of one or more instructions contained in main memory 706. Such instructions may be read into main memory 706 from another computer-readable medium, such as storage device 710. Execution of the sequences of instructions contained in main memory 706 causes processor 704 to perform the process steps described herein. One or more processors in a multi-processing arrangement may also be employed to execute the sequences of instructions contained in main memory 706. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0079] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 704 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 710. Volatile media includes dynamic memory, such as main memory 706. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 702. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[0080] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0081] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 704 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 700 can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector coupled to bus 702 can receive the data carried in the infrared signal and place the data on bus 702. Bus 702 carries the data to main memory 706, from which processor 704 retrieves and executes the instructions. The instructions received by main memory 706 may optionally be stored on storage device 710 either before or after execution by processor 704.

[0082] Communication interface 718 also provides a two-way data communication coupling to a network link 720 that is connected to a local network 722. For example, communication interface 718 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 718 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 718 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0083] Network link 720 typically provides data communication through one or more networks to other data devices. For example, network link 720 may provide a connection through local network 722 to a host computer 724 or to data equipment operated by an Internet Service Provider (ISP) 726. ISP 726 in turn provides data communication services through the worldwide packet data communication network now commonly referred to as the "Internet" 728. Local network 722 and Internet 728 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 720 and through communication interface 718, which carry the digital data to and from computer system 700, are exemplary forms of carrier waves transporting the information.

[0084] Computer system 700 can send messages and receive data, including program code, through the network(s), network link 720 and communication interface 718. In the Internet example, a server 730 might transmit a requested code for an application program through Internet 728, ISP 726, local network 722 and communication interface 718. In accordance with the invention, one such downloaded application provides for detecting RST-flood and/or SYN-RST denial of service attacks as described herein.

[0085] Processor 704 may execute the received code as it is received, and/or stored in storage device 710, or other non-volatile storage for later execution. In this manner, computer system 700 may obtain application code in the form of a carrier wave.

4.0 EXTENSIONS AND ALTERNATIVES

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.